HP Anywhere

Windows

Software Version: 10.00

Events Sample App Cookbook

Document Release Date: April 14, 2013

Software Release Date: March 2013

Legal Notices

Warranty

Restricted Rights Legend

Copyright Notice

Trademark Notices

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to:

http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Visit the HP Software Support Online web site at:

http://www.hp.com/go/hpsoftwaresupport

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

Contents

Chapter 1

Cookbook Overview

This document is a cookbook that describes the process of building and implementing the sample Events app using HP Anywhere capabilities.

Intended Audience

This document is intended for developers that use HP Anywhere APIs to build business applications that run within the HP Anywhere framework, and leverage HP Anywhere capabilities. This cookbook describes the development of a specific business application in detail.

It includes steps such as:

- The basic requirements needed to start app development
- How to read data from backend system
- Guides for implementation
- Specific HP Anywhere capabilities: participants, activities, recommended apps

After reading this guide, you should be able to implement a business application using the HP Anywhere APIs.

Prerequisites

Before using this cookbook, it is recommended that you familiarize yourself with:

- HP Anywhere User Guide, and related movies.
- HP Anywhere IDE Guide, and related movies.

Chapter 2

Events App Functional Description

The basic cookbook application is **Events**, which is a simple business application to manage events in your calendar.

The Events app functionality contains:

- **View Events list**, which is a simple list display for each event and contains the event name, event date, importance, and owner name. Events are sorted by date or name, according to user setting.
- **Create New Event**, which enables users to create an event that includes the following details:
  - **Date**. Default is the current date
  - **Name**. String, no default
  - **Importance**. High or regular.
  - **Owner (read-only)**. Automatically set according to user.
- **Delete Event**, which enables users to delete an event.
- **View/Edit Event**, which enables users to view or edit an event. Available from the landing page.
- **All Apps**, returns the View Events list and the New Event pages.
- **Recommended Apps**, returns the **View Event** page when a context object exists.
- **Recommended Participants**, returns an email, e.g., support@event.com, or something similar.
- **Collaboration on Events Page,** when not in an activity, automatically “inject” the event into activity context in the first post.
- **Automatic Creation of Activity,** when a new event is created (by the user), use the Handle Event API to automatically create an activity for the user, with the event as context object, and a message “A new event was created : <event name>”.
  a. The notification channel is “Front Page”.
  b. This functionality should be controlled by an admin setting - “create activity on new event” = “yes,no”

When a user clicks on the activity, the View Event entry point is opened.

Chapter 3

How to Open the Events App Code in HP Anywhere IDE

To open the code of the Events App in HP Anywhere IDE:

1. Download the <*Events_app*>.zip file from the Dev Zone.
2. Extract the source code from the zip file into your Eclipse workspace folder.
3. Open Eclipse.
4. From the **File** menu, select **Import > Existing Projects into Workspace** and click **Next.**
5. Click the **Browse** button next to **Select root directory**, and browse to the folder into which you extracted the source code.
6. Click **Finish**.

The Events app is now available for use from the Eclipse IDE Plugin.

Chapter 4

Recipes

This section describes the basic process to follow when developing apps and contains the following recipes:

- "How do I start writing the client side of an app?" on page 16
- "How do I get current user info?" on page 17
- "How do I provide a context object upon first collaboration?" on page 18
- "How do I save a state and reopen the activity later with this state?" on page 19
- "How do I define and use user settings?" on page 20
- "How do I adapt my app for different locales?" on page 21
- "How do I create a context object and prompt the user to add it to an activity?" on page 22
- "How do I start?" on next page
- "How do I read data from the backend system using REST?" on page 23
- "How do I create a new activity and send notifications (triggered by event)?" on page 28
- "How do I define and use admin settings?" on page 31
- "How do I implement All Apps (entry points)?" on page 32
- "How do I implement Recommended Apps (entry points)?" on page 33
- "How do I implement Recommended Participants?" on page 34

All the examples in the recipes are from the **Events** sample app.

How do I start?

This section describes the steps you need to take to implement your new app. At this stage, it is assumed you have already created the new Event app project and are ready to start the implementation.

There are some mandatory files, as well as optional classes and configuration files that you need to add to your project. These are described in the sections below:

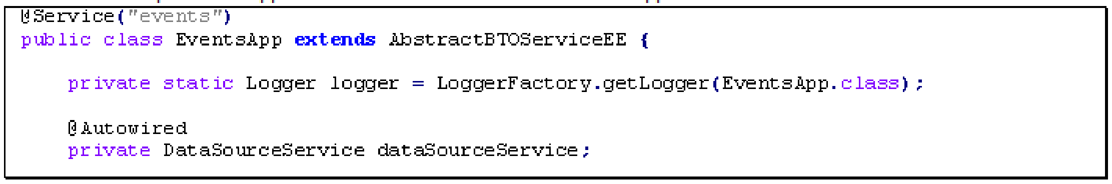
- Implement the Main App Service
- Add the App Descriptor
- Define the Data Source Provider
- Implement the REST Service for the App
- Add Localization Support

Implement the Main App Service (Mandatory)

To register an app in the HP Anywhere framework, each app must implement the **BaseBTOService** interface.

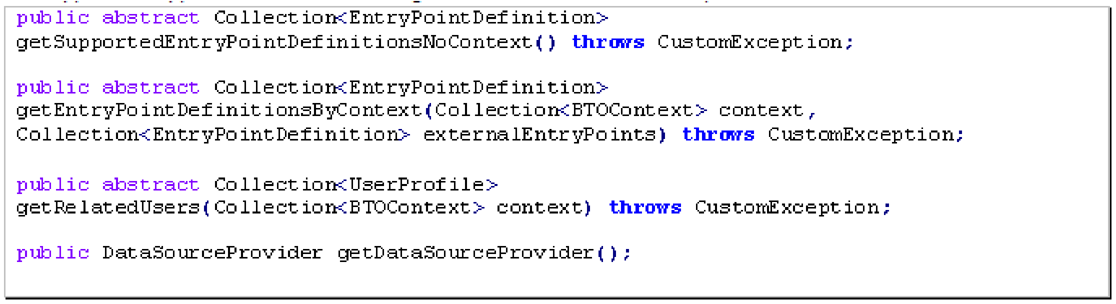
The HP Anywhere framework provides an abstract class AbstractBTOServiceEE that must be extended for the app service. The name of the Java Bean is the same as the app service ID that is defined in the descriptor.xml file.

In the example below, EventsApp extends AbstractBTOServiceEE and the app service ID is **Events**:



```
@Service("events")
public class EventsApp extends AbstractBTOServiceEE {

    private static Logger logger = LoggerFactory.getLogger(EventsApp.class);

    @Autowired
    private DataSourceService dataSourceService;
```

To support the app framework logic, the following four methods must be implemented:



```
public abstract Collection<EntryPointDefinition>
getSupportedEntryPointDefinitionsNoContext() throws CustomException;

public abstract Collection<EntryPointDefinition>
getEntryPointDefinitionsByContext(Collection<BTOContext> context,
Collection<EntryPointDefinition> externalEntryPoints) throws CustomException;

public abstract Collection<UserProfile>
getRelatedUsers(Collection<BTOContext> context) throws CustomException;

public DataSourceProvider getDataSourceProvider();
```

Add the App Descriptor (Mandatory)

The app metadata configuration file is mandatory. The app descriptor includes the basic app metadata such as the form factors the app support, the web resources relevant per form factor, the app ID and version.

The app descriptor is defined in the **<app-id>-descriptor.xml** file.

Note: app-id must be unique for each app.

This step, along with the step described in "Implement the Main App Service (Mandatory)" on previous page define a basic app.

For more details on the descriptor, see the *HP Anywhere API Reference*.

Define the Data Source Provider

A developer can specify how the app is connected to its legacy backend system. This is done by defining a data source provider. In most cases, the legacy backend is accessed via REST, so the connectivity definition includes properties such as host, port and protocol.

Note: There are apps that do not use a backend system, therefore this step is not mandatory.

There are two ways to implement the data source provider:

- Use the DataSourceProvider interface

  HP Anywhere provides a class named WSDataSourceProvider that can be extended. The developer must supply at least the data source name.

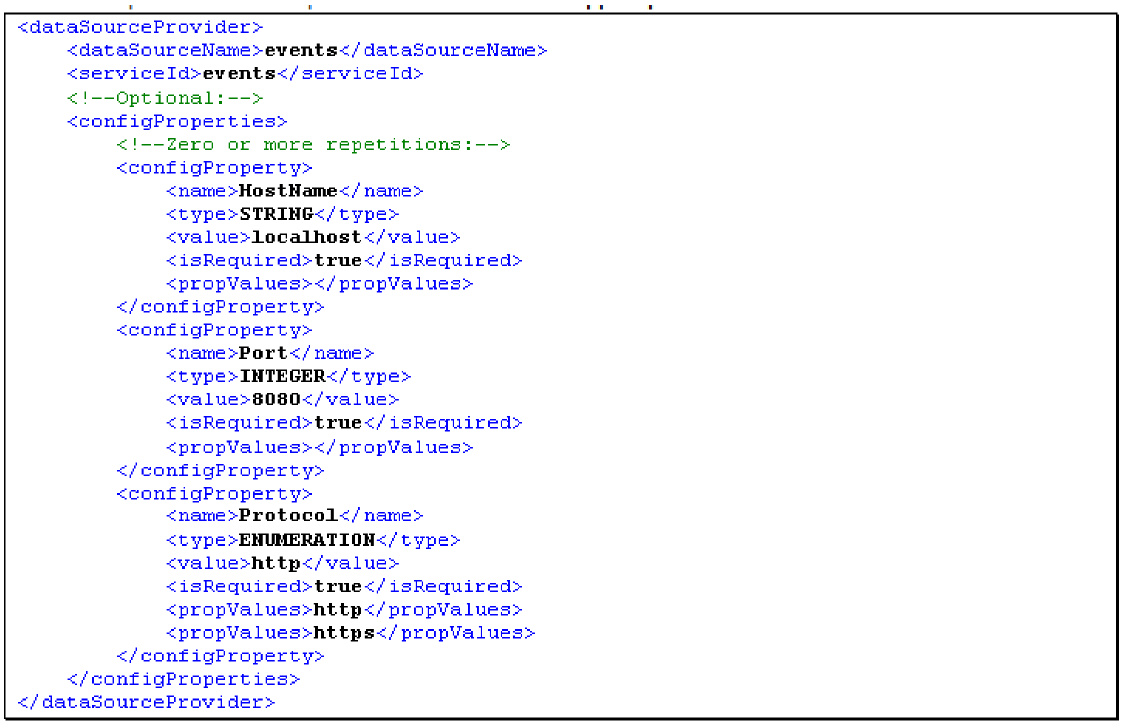
- Add the DataSourceProvider.xml file that includes the list of properties

  Upon deployment of the app on the HP Anywhere server, from the Administrator Console, its data source information is exposed. The admin user then configures the data source values from the Administrator Console, such as typing the host FQDN, port, and the protocol to access the backend system.

After the data source configuration is done, the app can compose the URL to its backend, using the DataSourceService that the HP Anywhere framework provides to get the data source information.

For details on the DataSourceProvider.xml, see the *HP Anywhere API Reference*.

In the Events app, the data source provider is defined in the **EventsApp-ds-provider.xml** file. In this file, the **dataSourceName** and **serviceId** parameters are mandatory. The remaining parameters are defined as required in the app.



```
<dataSourceProvider>
    <dataSourceName>events</dataSourceName>
    <serviceId>events</serviceId>
    <!--Optional:-->
    <configProperties>
        <!--Zero or more repetitions:-->
        <configProperty>
            <name>HostName</name>
            <type>STRING</type>
            <value>localhost</value>
            <isRequired>true</isRequired>
            <propValues></propValues>
        </configProperty>
        <configProperty>
            <name>Port</name>
            <type>INTEGER</type>
            <value>8080</value>
            <isRequired>true</isRequired>
            <propValues></propValues>
        </configProperty>
        <configProperty>
            <name>Protocol</name>
            <type>ENUMERATION</type>
            <value>http</value>
            <isRequired>true</isRequired>
            <propValues>http</propValues>
            <propValues>https</propValues>
        </configProperty>
    </configProperties>
</dataSourceProvider>
```

Each dataSourceProvider can define one or more configProperty elements. In the example above, in order to access the event legacy backend, you need a URL. Therefore, the properties required are protocol, host name and port.

Implement the REST Service for the App

For the app client to work with its legacy backend it generally includes a REST service that interacts directly with the backend. It is used for communication between the HP Anywhere server side and client side of the app. For details on HP Anywhere architecture, see the *HP Anywhere Administrator Guide*.

Example: In the following example, the REST service is responsible for supporting REST APIs that add, update and delete events, and the class **is EventsRestService**.

```
@Service
@Path("/events")
@Produces("application/json;charset=utf-8")
public class EventsRestService {

    @Autowired
    RestTemplate restClient;
    @Autowired
    LWSSOService securityService;
    @Autowired
    DataSourceService dataSourceService;
    @Autowired
    EventsApp eventsApp;
    @Autowired
    private UserInfoService userInfoService;
    @Autowired
    private AdminSettingsService adminSettingsService;
    .
    .
    .
    @GET
    public List<EventVo> getAll() {...}

    @GET
    @Path("{id}")
    public EventVo getEvent(@PathParam("id") String id) {...}

    @POST
    @Consumes("application/json;charset=utf-8")
    public EventVo addEvent(EventVo event) {...}

    @PUT
    @Path("{id}")
    @Consumes("application/json;charset=utf-8")
    public EventVo updateEvent(@PathParam("id") String id, EventVo eventVo)
{...}

    @DELETE
    @Path("{id}")
    public void deleteEvent(@PathParam("id") String id) {...}

    private String getBackendDatasourceBaseUrl() throws InvalidConfiguration-
Exception {...}

    private HttpEntity createRequestEntityBySessionCookie(MediaType medi-
aType){...}

    private HttpEntity createRequestEntityBySessionCookie(MediaType medi-
aType, Object requestObject){...}
    .
    .
    .
```

Add Localization Support

Each app must provide a resource bundle file, that includes a map of all localization keys with their translation per locale. The translation files name is composed of the **serviceId_locale.properties**.

Example: The **events_en.properties** example below displays the translation to the notification type properties in the admin setting xml. These translations appear in the Admin UI settings:

```
sections.notification.types.name=Notification Types

settings.notification.type.new.event.name=New Event

settings.notification.type.new.event.desc=Describe the notification channels
for new event
```

In addition, the HP Anywhere app framework provides the following methods to obtain the translated value of existing keys:

UserInfoService:getLocale	Find the locale of the current user session.
AbstractBTOServiceEE:getString	Retrieve the translated key.

In the following example, the getString API is used to retrieve the current locale.

```
String translatedKey = getString(userInfoService.getLocale(),"key.to.property.in.resource.bundle");
```

How do I start writing the client side of an app?

I'm developing the client side of a new app. How do I start?

You develop your app in JavaScript. The app runs on the user's device in an HTML iframe and communicates with HP Anywhere through a JavaScript API. API usage is explained throughout this document.

You do not need to define HTML markup for your app. Instead you declare your dependencies in a descriptor.xml file, and HP Anywhere injects them into your iframe. Dependencies may be JavaScript files, CSS files and so on. For performance reasons, it is highly recommended that you provide your app in a minified form. You can use any JavaScript framework (for example Sencha Touch 2, Enyo JS), CSS pre-processor (for example SCSS, LESS) and other tools, or none at all.

Your app must implement the top-level function **openEntryPoint (entryPointName, params, callback, scope)**. The first argument entryPointName is mandatory, **params**, **callback** and **scope** are optional. HP Anywhere calls this function whenever your app starts running.

A typical implementation of this function includes a switch statement on the given entry point, with a case clause for each possible entry point name. Your app must support a default entry point named **OpenEntryPoint**.

When your app is ready for user interaction, it must call **HPA.Interop.setReady()** with a single argument. The argument may simply be the boolean 'true', or an object indicating what capabilities the app has.

See Loader.js for the app's implementation of openEntryPoint().

How do I get current user info?

I want to use the current user info in my app, how can I get it?

You can retrieve the current user data by calling **HPA.Profile.getInfo()**. The returned object contains ID, display name, e-mail, job title, origin (internal/external) and several URLs for the user image in different sizes. In the example below, the display name is used.

Example

The following code in the **EventDetails.js** file retrieves the current user's display name, to set a new event owner:

```
HPA.Profile.getInfo().displayName
```

How do I provide a context object upon first collaboration?

Users of my app may start by working in the app, and then collaborate with others. How do I add the context objects they worked on to the current activity, only when they first collaborate?

When your app is loaded and you call the **setReady** function, pass an object with the property **hasAutoContext** set to true. On first collaboration, **openEntryPoint** is called and the value of the entryPointName parameter will be **getContextObject**. Also, accumulated context objects are available via the app's **getCtxObjects** function.

To do this:

1. Retrieve the context objects.
2. Pass them and your app to the callback using the passed scope.
3. Discard the accumulated context objects by calling s**etContextObject** for the next time the procedure is called.

Example

In **Loader.js**:

```
HPA.Interop.setReady({ hasDefaultAction: true, hasAutoContext: true});
```

In **Loader.js**, inside **openEntryPoint()**:

```
         case 'getContextObjects':
         {
             // get the current context from myApp;
             var ctxObjects = myApp.getCtxObjects();

             if (ctxObjects && ctxObjects.length > 0) {
                 // pass back the ctx objects
                 callback && enyo.isFunction(callback) &&
                 callback.apply(scope || this, [ ctxObjects,
                                                 EE.miniappId ]);

                 myApp.setCtxObjects([]);
             }
             return;
          }
```

How do I save a state and reopen the activity later with this state?

I want the app to open at the same location the next time I use the activity. How do I define this and when does it open?

A state may be any JavaScript value. The state is saved per activity, per user, and per entry point (app page). You may save a state at any time. When the user returns to an activity, the last saved state of each entry point is provided to that app page.

A state may include, for example, the context object the user last worked on, or the stage in a workflow the user last completed. In the example below, the state is the view the user was in (list or edit/new), and when performing an edit, the state is the data of the event last edited.

You save the state by calling **HPA.Interop.saveState()** with your state as the parameter.

You retrieve the state in the **openEntryPoint** implementation, from **params.state** (params is the second argument).

Example

The following examples show how to save and restore states in an app.

In the **EventContainer.js** file, inside **listEvents**:

```
HPA.Interop.saveState({ viewMode: 'list' });
```

In the **EventContainer.js** file, inside **editEvent**:

```
HPA.Interop.saveState({ viewMode: 'edit', data: data });
```

In the **Loader.js** file, inside **openEntryPoint()**:

```
if ( params.state && !!params.state.viewMode )
   {
     if ( params.state.viewMode === 'list' ){
            myApp.listEvents();
   }
     else if ( params.state.viewMode === 'edit' )
        {
            myApp.editEvent(null, state);
     }
  }
```

How do I define and use user settings?

How can I make specific settings available to my users, where each user has his/her own customized settings?

Locate the file **<App ID>-user-settings.xml** in your app's files. This file contains your app's user settings.

Edit this file using an XML editor (or text editor), and add your settings. Make sure you use the UserSettings.dtd file to verify that your XML file is valid and compliant. Some XML editors, such as the one built into the Eclipse IDE, provide a convenient way to build your XML from "compliant elements".

Each setting can be boolean, integer, enumeration and more. You can define default values as well as restrictions, for example, a maximum value.

For each setting, you define a name, a name key and a description key. These keys will be used to retrieve the name and description for a given user language and/or locale. You also provide keys in textual setting options. Settings are grouped into named contexts (sections).

To change their own settings, users can access the Settings section of HP Anywhere by clicking on ⚙.

In your app's xml file, define a section with the app name add the app settings in this section. Each type of setting may be rendered differently, for example a boolean setting may be rendered as a toggle button.

You access the settings of the current user through openEntryPoint's second argument, **params**. **Params.settings** contains your app's user settings with the values selected by the current user.

Example: Defining and using a user setting for events list ordering

In the **events-user-settings.xml** file, define:

```
<enumeration
        enum="events.sortSettings.sortBy.title,events.sortSettings.
               sortBy.date">
                events.sortSettings.sortBy.title
</enumeration>
```

In the **events_en.properties** file, define:

```
events.sortSettings.sortBy.title.name.key=Event Name
events.sortSettings.sortBy.date.name.key=Event Date
```

In the **Loader.js** file, define:

```
var sortOrderSettingValue = params.settings['events.sortSettings']
                                ['events.sortSettings.sortBy'][0];
```

How do I adapt my app for different locales?

I want my app to display user messages according to the current user's locale. How can I provide different text for each locale?

Under the **l10n** folder, create a file for each supported locale with the file name <App ID>_<locale code>.properties. For example, events_en.properties for English texts, events_de.properties for German, and so on. The keys are identical in all files, and the values of the text parameters are in the appropriate language.

HP Anywhere loads only the single file for the current user locale.

You access the value for a given key using **HPA.I18n.localize('key')**.

Example: Defining a key value and using the key to retrieve the value

In the **events_en.properties** file, define:

```
events.listTitle=My Events
```

In the **events_de.properties** file, define:

```
events.listTitle=Meine Ereignisse
```

In the **EventsList.js** file, define:

```
HPA.I18n.localize('events.listTitle') // Returns 'My Events' or
          'Meine Ereignisse', depending on user locale.
```

How do I create a context object and prompt the user to add it to an activity?

My app just generated a context object. How do I prompt the user to add this context object to the current activity or add it to a new activity?

1. Create a context object by calling **HPA.Interop.createContextObject()**.
2. Pass your object's ID, data type, display name and metaData (optional) as parameters to this function. (All these parameters are strings.)

   HPA.Interop.createContextObject() returns a JavaScript object if it created a valid contextObject, otherwise null.
3. If it created a valid context object, you can add it to an activity by calling **HPA.Interop.addContextObjectToExperience()** function, passing your object as a parameter.

The user will be prompted to select whether to add the object to the current activity or add it to a new activity. The user can also dismiss the prompt without selecting an option.

Example: Share an event by creating a context object and adding it to an activity

In the **EventContainer.js** file, inside **shareEvent()**:

```
// create new ctx object via HPA.Interop.createContextObject
     factory method
var ctx = HPA.Interop.createContextObject(newData.id, "EVENT",
      newData.title, JSON.stringify(newData));
// share the newly created ctx object
ctx && HPA.Interop.addContextObjectToExperience(ctx);
```

How do I read data from the backend system using REST?

In the Events app, the backend can be accessed using the REST APIs. Using these APIs, you can:

- View all events
- View event (view a single event)
- Create a new event
- Update an existing event
- Delete an event

The backend REST server is represented by the **LegacyEventsEmulatorService** class. This class is added to the app as an emulator for a real backend system.

The LegacyEventsEmulatorService exposes the following REST APIs:

1. View all events

Method	GET
URL	http://backend.fqdn:8080/events-web/services/backendLegacyEventsEmulator
Body	Empty

2. View event

Method	GET
URL	http://backend.fqdn:8080/events-web/services/backendLegacyEventsEmulator/{id}
Body	Empty

3. Create new event

Method	POST
URL	http://backend.fqdn:8080/events-web/services/backendLegacyEventsEmulator
Body	JSON representation of EventVo

For example:

```
{
"id":"1362474882964",
"owner":"admin admin",
"date":"2013-01-01",
"importance":"LOW",
"title":"my event6"
}
```

4. Update existing event

Method	PUT
URL	http://backend.fqdn:8080/events-web/services/backendLegacyEventsEmulator/{id}
Body	JSON representation of EventVo

For example:

```
{
"id":"1362474882964",
"owner":"admin admin",
"date":"2013-01-01",
"importance":"LOW",
"title":"my event6"
}
```

5. Delete event

Method	DELETE
URL	http://backend.fqdn:8080/events-web/services/backendLegacyEventsEmulator/{id}
Body	empty

Reading Data from the Backend Legacy Server

To read data from the backend legacy server, follow the steps below:

"1. Get the Events app data source information to create backend base URL"

"2. Implement the REST request URL to access the relevant resource in the backend"

" 3. Get the session security cookie for backend authorization"

"4. Invoke the REST call and analyze the return value"

1. Get the Events app data source information to create backend base URL

The data source information is managed by the administrator via the Admin UI.

Note: The data source provider must be implemented, see " Define the Data Source Provider" on page 12 above.

To retrieve the data source information, the HP Anywhere framework provides the DataSourceService, which includes the following getDataSourceConfig method:

```
public DSConfiguration getDataSourceConfig(String dataSourceName,String
instanceName)
```

where:

dataSourceName is the name of the data source to retrieve, usually the same as the app service ID.

instanceName is the name of the data source instance to retrieve. if null, method returns the first data source instance for the App.

This method returns DSConfiguration which is a key-value map of data source for the current app. The keys are according to the data source provider xml that the app provided.

The following example displays how to build the base URL in events app using the DataSourceService:

```
private String getBackendDatasourceBaseUrl() throws
InvalidConfigurationException {
    //retrieve data source configuration from the HPA server
    DSConfiguration dataSource =
dataSourceService.getDataSourceConfig(eventsApp.getID(), null);

    //build the base url to access the backend according to the value of
known data source properties
    String protocol =
(String)dataSource.getPropertyValue(DataSourceConsts.PROTOCOL_KEY_NAME);
    String hostname =
(String)dataSource.getPropertyValue(DataSourceConsts.HOSTNAME_KEY_NAME);
    String port =
(String)dataSource.getPropertyValue(DataSourceConsts.PORT_KEY_NAME);

    String requestUrl = protocol + "://" + hostname + ":" + port+ "/" ;

    return requestUrl;
}
```

Use the DSConfiguration **getPropertyValue** method to retrieve the data source properties values.

2. Implement the REST request URL to access the relevant resource in the backend

Update the request URL to access the relevant resource.

To access the backend REST Server use Spring's RestTemplate. RestTemplate is a Spring central class for client-side HTTP access. It simplifies communication with HTTP servers, and enforces RESTful principles. It handles HTTP connections, leaving application code to provide URLs (with possible template variables) and extract results.

To use RestTemplate, all you need to do is to inject RestTemplate bean to your REST Service class, for example:

```
@Service
@Path("/events")
@Produces("application/json;charset=utf-8")
public class EventsRestService {

    @Autowired
    RestTemplate restClient;
```

After you have the backend base URL, append the relevant resource path. For example:

```
@GET
public List<EventVo> getAll() {

    List<EventVo> events = new ArrayList<EventVo>();
    String requestUrl = null;

    try {
        requestUrl = getBackendDatasourceBaseUrl()+ SERVICE_SUFFIX_PATH ;
    } catch (InvalidConfigurationException e) {
        logger.error("Failed to retrieve data source configuration details,
with exception:" + e);
        //error occurred, returning empty events list
        return events;
    }
.
.
.
```

3. Get the session security cookie for backend authorization

In order to build the HTTP request, use the private method in the EventsRestService class.

```
private HttpEntity createRequestEntityBySessionCookie(MediaType mediaType,
Object requestObject)
```

1. Use LWSSOService to retrieve the security cookie. For example:

```
        Cookie sessionCookie = null;

        try {
            sessionCookie =
        securityService.getSecurityCookie(userInfoService.getUserName());
            logger.debug("Success to get cookie from session , cookie name : " +
        sessionCookie.getName() + " , Cookie value: " + sessionCookie.getValue());
        } catch (Exception e) {
            logger.error("failed to retrieve session cookie", e);
        }
```

2. Use **UserInfoService getUserName** to retrieve the user name of the user currently logged in.
3. Add relevant headers, cookie and body to pass them to the request.

   In this example the server accepts JSON media type and also requires CSRF header:

```
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_JSON);
headers.add("X-CSRF-HPMEAP","FROM-EVENTS-APP");
```

4. Add the security session cookie that is retrieved earlier to the HttpHeaders:

```
if (sessionCookie != null) {
    headers.add("Cookie", sessionCookie.getName() + "=" +
sessionCookie.getValue() + ";" + sessionCookie);
}else{
    logger.error("session cookie is missing, return requestEntity without
cookie");
}
```

5. Create request HttpEntity with the headers and request body object:

```
HttpEntity request;
if (requestObject == null){
    request = new HttpEntity(headers);
}else{
    request = new HttpEntity(requestObject, headers);
}

return request;
```

4. Invoke the REST call and analyze the return value

To invoke the REST call, use the REST template's exchange method.

For example, follow the exchange method on the Delete action:

```
@DELETE
@Path("{id}")
public void deleteEvent(@PathParam("id") String id) {
    String requestUrl = null;

    try {
        requestUrl = getBackendDatasourceBaseUrl() + SERVICE_SUFFIX_PATH +
"/{id}";
    } catch (InvalidConfigurationException e) {
        logger.error("Failed to retrieve data source configuration details,
with exception:" + e);
    }

    HttpEntity requestEntity =
createRequestEntityBySessionCookie(MediaType.APPLICATION_JSON);


    ResponseEntity<EventVo> response = null;
    Map<String, String> uriParams = new HashMap<String, String>();
    uriParams.put("id", id);

    try {
        response = restClient.exchange(requestUrl, HttpMethod.DELETE,
requestEntity, EventVo.class,uriParams);
    } catch (RestClientException e) {
        logger.warn("delete event failed with RestClientException", e);
    }

    if (response !=null ){
        if (response.getStatusCode() == HttpStatus.OK){
            logger.debug("received successful response status ");
        }else{
            logger.error("response status is:" + response.getStatusCode());
        }
    }
}
```

In the code example above, you compose the requestUrl that is the URL to the legacy backend server resource, then build the requestEntity that is HttpEntity that includes the relevant headers and the session cookie.

You then invoke the REST call to the backend by using Spring RestTemplate exchange method:

restClient.exchange(requestUrl, HttpMethod.DELETE, requestEntity, EventVo.class, uriParams)

From this point, the app can perform its business logic according to the response.

How do I create a new activity and send notifications (triggered by event)?

There are cases in which an app needs to create new activity and send notification to the activity's participants. In other cases the app needs to send notifications about an existing activity. This can be done by triggering an event with a context object using the Handle Event REST API.

For details, see the *HP Anywhere API Reference*.

In the Events app example, to demonstrate the handle event feature, invoke the event REST call when there is a new event added by the user.

URL	http://localhost:8080/diamond/rest/api/V2/apps/{appId}/events
Method	POST
URI params	appId = events

Body	`{`
	`"id": "eventId1",`
	`"contextObjects": [`
	`{`
	`"objectId": "135893211785089",`
	`"dataType": "event",`
	`"displayName": "Calendar Event",`
	`"metaData": "user_1@hp.com",`
	`"app": {`
	`"id": "events"`
	`}`
	`}`
	`],`
	`"message": "New activity created, by triggering app event",`
	`"users": [`
	`{ "id": "user_1@hp.com",`
	`"isNotify": "true"`
	`},`
	`{ "id": "user_2@hp.com",`
	`"isNotify": "true"`
	`}`
	`],`
	`"visibility": "PRIVATE",`
	`"notificationType": "new.event.notification",`
	`"subject": "Event Activity Subject"`
	`}`

In this example, the app sends an event in order to create new activity.

By invoking the REST API, the following actions occur:

1. A new activity is created with the subject: "Event Activity Subject"
2. A system post is added to the activity timeline: "New activity created, by triggering app event"
3. The activity includes the following participants: user_1@hp.com, user_2@hp.com
4. The visibility of the activity is set as PRIVATE, which means only a participant in the activity

can see it and search for it.

5. A context object of type event and id 135893211785089 is added to the activity.

Send notification

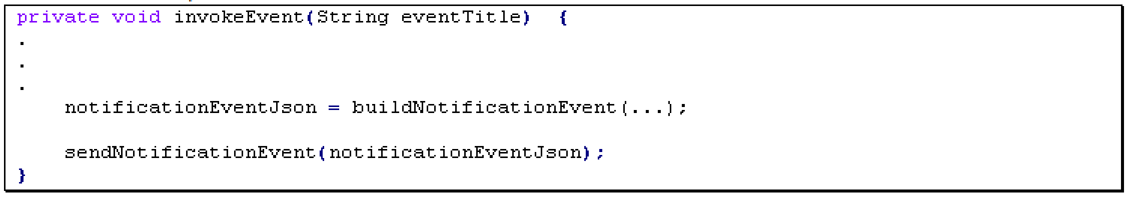
In addition to the activity creation, the app can send a notification. push or email messages to several channels as defined in the app admin settings. This is part of the Events REST API.

To send a notification you must:

- " Invoke the Event notification REST API"
- "Define notification type in the Admin Setting xml"

Invoke the Event notification REST API

See the code sample below, that invokes the Event REST API:

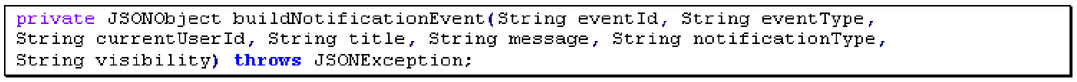


```
private void invokeEvent(String eventTitle)  {
.
.
.
    notificationEventJson = buildNotificationEvent(...);

    sendNotificationEvent(notificationEventJson);
}
```

To invoke a notification event, you must:

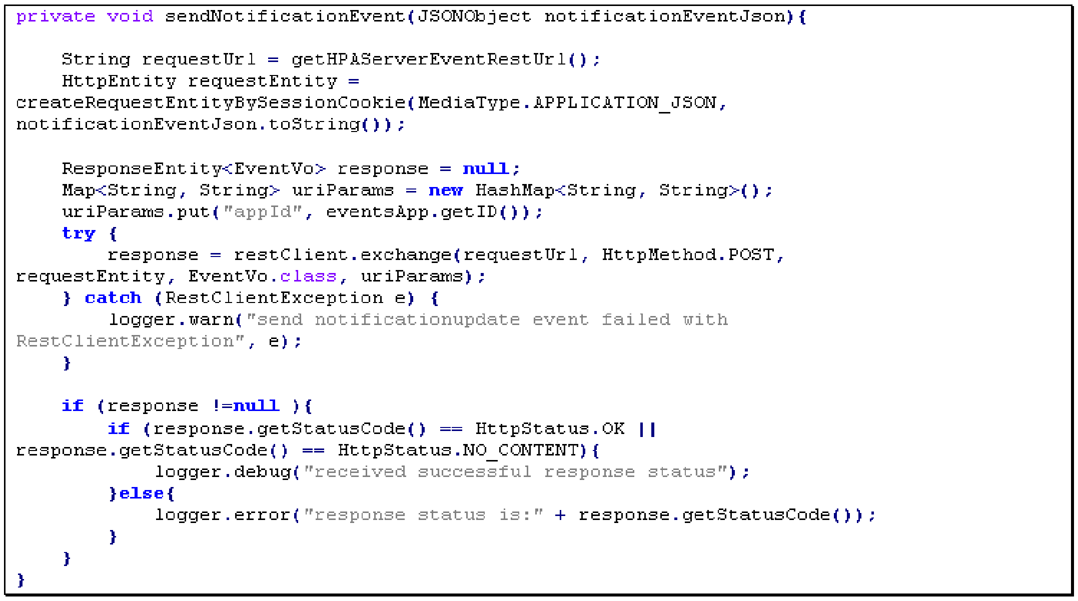
1. Compose the JSON body for the event with the relevant values.
2. Send the event:
   a. Compose the HP Anywhere server URL with the event REST API to invoke.
   b. Invoke the REST API using the REST client (similar flow to invoking backend REST API).

   In the following example, in order to compose the JSON body for the event, use EventRestService:buildNotificationEvent method:



```
private JSONObject buildNotificationEvent(String eventId, String eventType,
String currentUserId, String title, String message, String notificationType,
String visibility) throws JSONException;
```

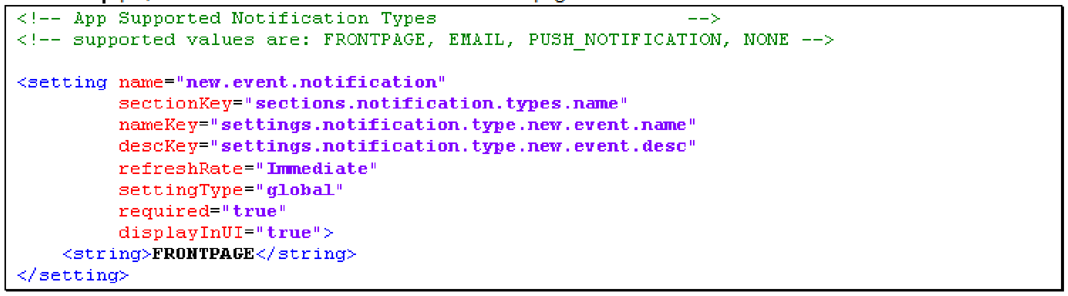
   Finally, here is how we actually invoke REST API using the REST client (similar flow to invoking backend REST API):



```
private void sendNotificationEvent(JSONObject notificationEventJson){

    String requestUrl = getHPAServerEventRestUrl();
    HttpEntity requestEntity =
createRequestEntityBySessionCookie(MediaType.APPLICATION_JSON,
notificationEventJson.toString());

    ResponseEntity<EventVo> response = null;
    Map<String, String> uriParams = new HashMap<String, String>();
    uriParams.put("appId", eventsApp.getID());
    try {
        response = restClient.exchange(requestUrl, HttpMethod.POST,
requestEntity, EventVo.class, uriParams);
    } catch (RestClientException e) {
        logger.warn("send notificationupdate event failed with
RestClientException", e);
    }

    if (response !=null ){
        if (response.getStatusCode() == HttpStatus.OK ||
response.getStatusCode() == HttpStatus.NO_CONTENT){
            logger.debug("received successful response status");
        }else{
            logger.error("response status is:" + response.getStatusCode());
        }
    }
}
```

Define notification type in the Admin Setting xml

The notification channels are defined for the “new.event.notification”. Enter the notification type in the <string> </string> parameter. This may be FRONTPAGE, PUSH_NOTIFICATION, EMAIL or NONE. In this example, the notification is sent to the HP Anywhere Front Page:



```
<!-- App Supported Notification Types                      -->
<!-- supported values are: FRONTPAGE, EMAIL, PUSH_NOTIFICATION, NONE -->

<setting name="new.event.notification"
         sectionKey="sections.notification.types.name"
         nameKey="settings.notification.type.new.event.name"
         descKey="settings.notification.type.new.event.desc"
         refreshRate="Immediate"
         settingType="global"
         required="true"
         displayInUI="true">
    <string>FRONTPAGE</string>
</setting>
```

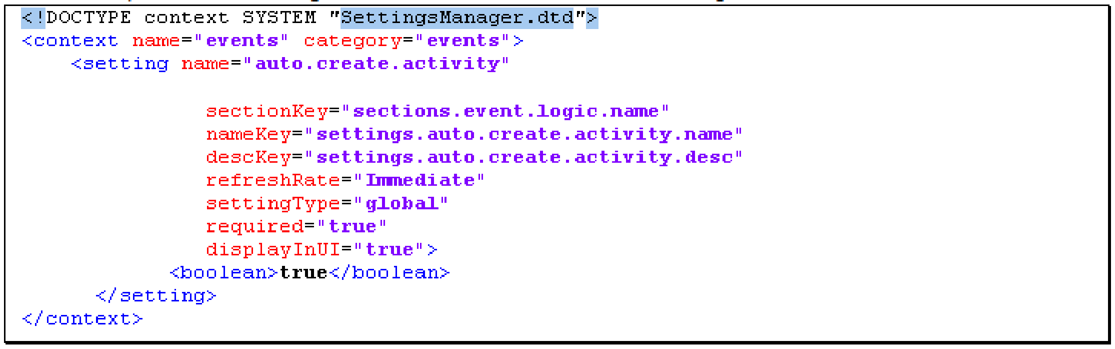
How do I define and use admin settings?

There are cases in which an app requires configurations that impact the app behavior of the app. You can provide a specific Admin Settings XML file for your app that includes these configuration. Upon deployment of the app, the app settings are exposed in the admin console, and the admin user can then configure its values.

The app can then run these values in runtime, using the AdminSettingService of the HP Anywhere framework.

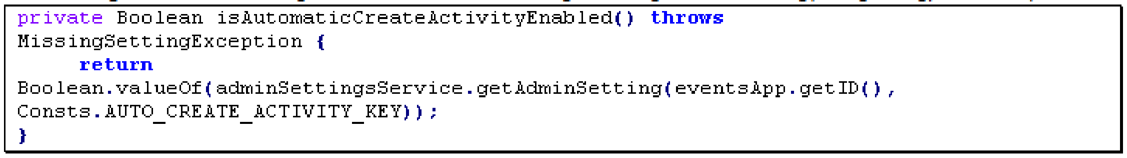
For more details on the admin settings see the *HP Anywhere API Reference*.

In the example the admin setting XML is defined in events-admin-settings.xml.



```
<!DOCTYPE context SYSTEM "SettingsManager.dtd">
<context name="events" category="events">
    <setting name="auto.create.activity"

               sectionKey="sections.event.logic.name"
               nameKey="settings.auto.create.activity.name"
               descKey="settings.auto.create.activity.desc"
               refreshRate="Immediate"
               settingType="global"
               required="true"
               displayInUI="true">
            <boolean>true</boolean>
      </setting>
</context>
```

In the event app example, use the admin setting to decide whether the app needs to create new activity on add event. To get the admin setting value, use the AdminSettingService: getAdminSetting(String,String), for example:



```
private Boolean isAutomaticCreateActivityEnabled() throws
MissingSettingException {
     return
Boolean.valueOf(adminSettingsService.getAdminSetting(eventsApp.getID(),
Consts.AUTO_CREATE_ACTIVITY_KEY));
}
```

How do I implement All Apps (entry points)?

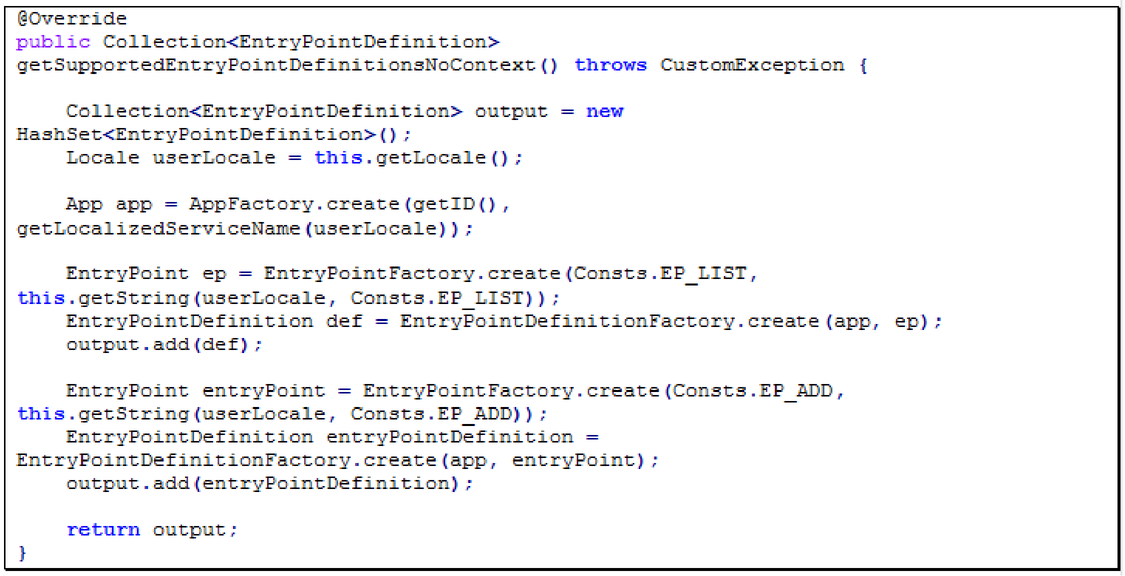
In order to implement All Apps entry points, you need to implement the method:

AbstractBTOServiceEE:**getSupportedEntryPointDefinitionsNoContext**

This method is called by the HP Anywhere framework, in order to show all available entry points in your app that can be opened without a given context object.

In the Events app example, two entry points without context are supported: View Events list and Add Event.

In the example below from the EventApp class, notice how we use the factories to create the list of the app-supported entry points without context:



```
@Override
public Collection<EntryPointDefinition>
getSupportedEntryPointDefinitionsNoContext() throws CustomException {

    Collection<EntryPointDefinition> output = new
HashSet<EntryPointDefinition>();
    Locale userLocale = this.getLocale();

    App app = AppFactory.create(getID(),
getLocalizedServiceName(userLocale));

    EntryPoint ep = EntryPointFactory.create(Consts.EP_LIST,
this.getString(userLocale, Consts.EP_LIST));
    EntryPointDefinition def = EntryPointDefinitionFactory.create(app, ep);
    output.add(def);

    EntryPoint entryPoint = EntryPointFactory.create(Consts.EP_ADD,
this.getString(userLocale, Consts.EP_ADD));
    EntryPointDefinition entryPointDefinition =
EntryPointDefinitionFactory.create(app, entryPoint);
    output.add(entryPointDefinition);

    return output;
}
```

EntryPointDefinition consists of entryPoint and its owner app. Each object consists of an id and name.

The name will be displayed in the user interface, therefore it should be localized.

How do I implement Recommended Apps (entry points)?

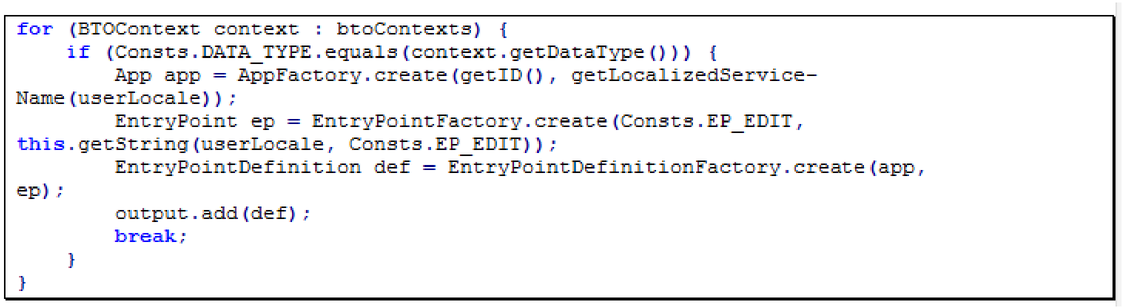
In order to implement recommended apps feature, you need to implement the method:

AbstractBTOServiceEE: **getEntryPointDefinitionsByContext**

This method is called by the HP Anywhere framework, with a list of context objects and opened entry points that exist in current activity. According to these lists, the app should perform its own business logic and return recommended entry points.

In the Events app example, we recommend the Edit entry point if a context exists with a type that is one of the supported types.

In the example below from the EventApp class, notice how we validate the context object to apply our own recommendation logic:



```
for (BTOContext context : btoContexts) {
    if (Consts.DATA_TYPE.equals(context.getDataType())) {
        App app = AppFactory.create(getID(), getLocalizedService-
Name(userLocale));
        EntryPoint ep = EntryPointFactory.create(Consts.EP_EDIT,
this.getString(userLocale, Consts.EP_EDIT));
        EntryPointDefinition def = EntryPointDefinitionFactory.create(app,
ep);
        output.add(def);
        break;
    }
}
```

A similar logic of checking the list of the context objects can be applied to the list of open entry points where entry points are recommended according to what is currently opened.

How do I implement Recommended Participants?

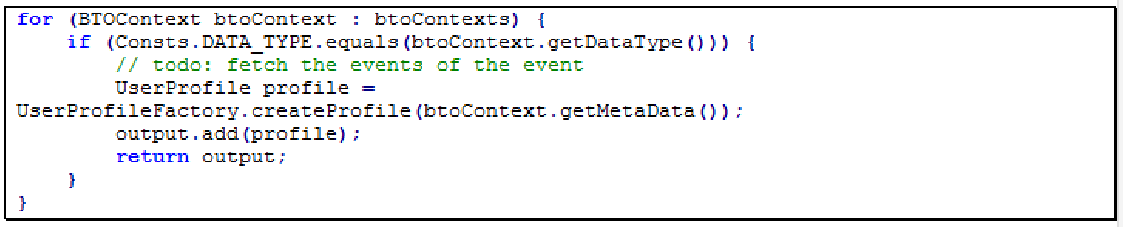
To support recommended participants, the app should return a list of user ids (as defined in the user repository).

In order to implement the recommended participants feature, you need to implement the method:

AbstractBTOServiceEE: **getRelatedUsers**

This method is called by the HP Anywhere framework, with a list of context objects and open entry points that exist in the current activity. According to these lists, the app should perform its own business logic and return a list of the recommended participants.

The following example is from the EventApp class. Notice how **UserProfileFactory.createProfile** is used to compose the list of recommended users:



```
for (BTOContext btoContext : btoContexts) {
    if (Consts.DATA_TYPE.equals(btoContext.getDataType())) {
        // todo: fetch the events of the event
        UserProfile profile =
UserProfileFactory.createProfile(btoContext.getMetaData());
        output.add(profile);
        return output;
    }
}
```

In the EventApp example, when the app saves the context object, it also saves possible recommended users in the context object metadata field. When the app is called, it retrieves the recommended users from the context object.

This method was used in order to save time of retrieving the relevant users from the backend according to context Id.

Note that you can implement a different method for producing recommend users, by accessing the backend data source, or recommending according to the given list of open entry points.

Java™
COMPATIBLE